AFRL-IF-RS-TR-2002-278
Final Technical Report
October 2002

AGENTWARE: AUTOMATED SYNTHESIS OF SOFTWARE AGENTS

Kestrel Institute

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J389 & G347

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-278 has been reviewed and is approved for publication

John J. Lemmer

APPROVED:

JOHN LEMMER
Project Engineer

Michael Talbert

FOR THE DIRECTOR:

MICHAEL L. TALBERT, Maj., USAF
Technical Advisor, Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** October 2002	**3. REPORT TYPE AND DATES COVERED** Final Jun 98 – Sep 01

4. TITLE AND SUBTITLE
AGENTWARE: AUTOMATED SYNTHESIS OF SOFTWARE AGENTS

5. FUNDING NUMBERS
C - F30602-98-C-0169
PE - 63760E
PR - AGEN
TA - T0
WU - 12

6. AUTHOR(S)
Douglas R. Smith and Stephen J. Westfold

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Kestrel Institute
3260 Hillview Avenue
Palo Alto California 94304

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFTB
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2002-278

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: John Lemmer/IFTB/(315) 330-3657/ John.Lemmer@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*

This report describes our research on developing and applying synthesis technology to agent-based systems in the DARPA/AFRL COABS program. We summarize our results in the following areas: generic synthesis frameworks, synthesis of scheduling agents, synthesis of authentication protocols, formal metalevel specifications, and synthesis of authentication protocols, formal metalevel specifications, and synthesis of glue code.

14. SUBJECT TERMS
Agents, Synthesis, Formal Specifications, Glue Code, Scheduling, Protocols

15. NUMBER OF PAGES
16

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

1. Executive Summary

This report describes our research on developing and applying synthesis technology to agent-based systems in the DARPA/AFRL Control of Agent-Based Systems (COABS) program. Our technical approach is based on *specification refinement* technology which allows the systematic machine-supported development of software from requirement specifications. The refinements embody programming knowledge about algorithms, data structures, program optimization techniques, etc. The result of the refinement process is executable code that is consistent with the problem specification. The development process can produce highly efficient code along with a proof of the code's correctness.

The initial goal of the project was to develop generic tools to support the construction of software agents software, in particular agents that provide scheduling and resource allocation services. In the second part of the project, we refocused on a critical aspect of coordinating the interaction of agents: the synthesis of glue-code to enable agents to communicate even when they expect data in different formats and at different levels of abstraction. We also explored the synthesis of authentication protocols via composition mechanisms.

We obtained technical results in the following areas. Alongside each topic area, we list the names of systems that we built to implement these results.

1. Generic Synthesis Frameworks (Designware)
2. Synthesis of Scheduling Agents (Planware)
3. Synthesis of Authentication Protocols
4. Formal Metalevel Specifications (leading to MetaSlang)
5. Synthesis of Glue Code (glue-code generator, MIATA TIE contributions)

Our technical results, detailed examples, and discussions of implemented systems are documented in 11 publications that grew out of the work on this project.

2. Introduction

This final report summarizes the work performed by Kestrel Institute on the project ``Agentware: Automated Synthesis of Software Agents", Contract No. F30602-98-C-0169 under the DARPA/AFRL Control of Agent-Based Systems (COABS) Program. The project ran from 30 June 1998 through 30 September 2001.

The initial goal of this project was to develop generic tools to support the construction of software agents software. Our technical approach is based on *specification refinement* technology which allows the systematic machine-supported development of software from requirement specifications. The refinements embody programming knowledge about algorithms, data structures, program optimization techniques, etc. The result of the refinement process is executable code that is consistent with the problem specification. The development process can produce highly efficient code along with a proof of the code's correctness. The proposal and early part of the project focused on technology for synthesizing software agents, in particular agents that provide scheduling and resource allocation services.

In the second part of the project (in coordination with DARPA and AFRL), we focused on a critical aspect of coordinating the interaction of agents: the synthesis of glue-code to enable agents to communicate even when they expect data in different formats and at different levels of abstraction. We also explored the synthesis of authentication protocols via composition mechanisms.

This report is structured as follows. Section 2 presents an overview of technical results obtained during this project. Section 3 presents our results on synthesis of glue-code in more detail. Section 4 lists the publications that resulted from this project.

3. Overview of Technical Results

3.1. Synthesis of Scheduling Agents

Our goals for this project were to explore ideas in synthesizing software agents, and to implement those ideas in an extension of Specware/Designware/Planware systems [18],[16], [1]. We laid out the following tasks.

1. Develop Designware infrastructure
    1.1. Diagram colimit - algorithm, interface
    1.2. Taxonomy support at interface
    1.3. Ladder construction interface
    1.4. Interpretation construction interface
        - propagation rules
        - support for unskolemization
        - support for manual definition
        - for connections and other specialized construction methods
2. Develop a taxonomy of agent architectures
3. Synthesize a scheduling agent
    - finish CP tactic

- program optimization rules (CD-simplify)
- spreadsheet interface development

We made considerable progress in the development of the Designware framework, as reported in [16]. We accomplished Tasks 1.1 through 1.4 above, and carried out a variety of example derivations. We faced and solved a number of technical difficulties along the way. One technical problem concerned the use of diagrams of specifications (i.e. structured specifications) and their colimits (composition of structured specifications). In essence, our first implementation did not allow certain equalities between paths to be preserved in the colimit. Technically this is the problem of allowing nonfree shape categories underlying the diagrams. We solved this problem and implemented an extension to the diagram operations to allow nonfree shapes. Although quite technical, this result is necessary that the colimit give the kind of result expected in synthesis applications.

We also made considerable progress on Task 3 above (Synthesize a scheduling agent). Our goal was to create a scheduler vending agent on the net. An agent that requires scheduling services interacts with the scheduler vendor, supplying the necessary problem-specific detail, and then, after appropriate payment, receives a scheduling agent together with the necessary data translators (see Section 3.4) and an appropriate GUI. The synthesis of GUI's would probably be necessary for a successful vendor, but it wasn't a focus of this project.

The key to creating this vending agent was to make the acquisition of scheduling problem-specific detail as simple and uniform as possible. One key insight was our discovery that all of the constraints that characterize the schedulers that we have developed in recent years have a common abstract form, technically they are definite constraints over a semilattice [15],[20]. This enabled us to set up a spreadsheet-like interface for acquiring the essential information about scheduling constraints. The formalism allows Planware to automatically convert the spreadsheet entries into detailed logical constraints without the purchasing agent/user needing to know logic or category theory.

We reworked the abstract scheduling spec (from Planware) to allow the choice of resources from a taxonomic library, and to generate task refinements based on input from the spreadsheet-like interface. We developed a parser/linker function to convert spreadsheet text formulas into the internal format used by the underlying Specware system.

We developed an initial version of the spreadsheet interface in the scheduler generator. The interface allows users to modify the default entries in the spreadsheet, and the system will parse the results, and create appropriate task attributes and semilattices, and finally a formal specification of the user's scheduling problem. This allows us to generate large specifications (about 50 pages of text) from a simple table of bounding information supplied by the user. In previous work with KIDS, the user had to write this large specification by hand prior to performing synthesis.

We delivered the newest version of Planware, including the spreadsheet interface, to AFRL in June 1999 for use as a demo system.

3.2. Protocol Synthesis

The project supported a low-level collaboration with Prof. John Mitchell (Stanford University) on techniques for generating correct authentication protocols between agents. We founded our formalization of protocol composition on the concept of strand spaces, developed at MITRE. A strand is a sequence of events, usually communication events, and they are connected to one another to build larger strands, ultimately a complete protocol. This gives a clear foundation for protocol composition. We deepened that formalism by defining a strand category with arrows giving the interconnections of strands. We developed a logic for specifying and reasoning about properties of strands [5], [4].

We were able to sketch out the composition of a simple protocol, the famous Needham-Schroeder-Lowe authentication protocol. A first version of this protocol (called Needham-Schroeder) was published 25 years ago and was found to be flawed after 15 years of use, despite ``proofs" of its correctness. Our approach yields the corrected Needham-Schroeder-Lowe protocol by composition.

3.3. Meta-theories

We began work on a meta-theory of Specware specs to capture notions of expression optimization and architecture in a general way in Designware. This is crucial foundational work for synthesis of software agents because agent architectures require specification at the meta-level and, more generally, alot of software design knowledge is best expressed at the meta-level. Architectures are defined in terms of components, connectors, and system invariants. The component interfaces are specified (in Specware) via formal specifications. The architectural structure of component interconnections must be at the meta-level. The technical report [17] helped motivate the redesign of the Specware language and resulted in the MetaSlang system which is the current foundation of all work at Kestrel.

3.4. Synthesis of Glue Code

Through discussions with MIATA TIE group we refocused our effort on glue code for a route generator. The idea was to treat the AMC CAMPS Mission Planner (that we had co-developed previously with BBN) as an agent that requests routing services, and moreover, to treat the route generator as requiring detailed flight duration and flight path services for a given aircraft, flight leg, and departure time. A great-circle-route agent and CIRL's WARP would provide alternate agents for providing these services. More generally, we sought to explore the formal derivation of glue code that translates between the data offered by one agent and the required data of another. We worked a number of example problems drawn from the CAMPS airlift scheduling domain[6].

We made significant progress on formally deriving glue code. Here are the key ideas: Given agent A that produces data source S, and agent B that requires data T, we want to derive glue code f such that f (S) = T. We first need to reconcile the semantics of agents A and B by producing a common abstract domain theory T. The derivation of f takes place in the theory formed by unioning the theories of A and B modulo the common theory T -- technically this is a colimit operation. Given this setting, we found we could readily derive f by interleaving the basic steps of a higher-order matching algorithm with application of domain-specific theorems as necessary. The result is a data translator that is correct-by-construction.

As a typical example, we are given a scheduling agent MP that produces an airlift schedule -- for each aircraft, the schedule gives the sequences of flights that it makes. We also have an agent CM that requires what is known as a commitment matrix -- the number and type of aircraft that are committed (i.e. not free for allocation) over time. The problem is to derive a translator *f* from schedules to commitment matrices. This problem has features of translation and summarization of data.

There are two key steps in formalizing the problem so that it admits a rigorous and general solution method. First, there must be a shared language/theory in which both MP and CM can be described and their shared ontology made explicit. One approach is to develop an abstract common theory of the domain, which is airlift scheduling (AS) here. Next, theories for MP and CM are developed as extensions of AS where the schedule and commitment matrix datatypes are expressed in terms of the language of AS. The problem of translating from schedule to commitment matrix datatypes is expressed in the pushout (shared union) of these theories.

Second, the translation problem can be treated as solving a higher-order equation. For example, if S is a term constructing an MP schedule and T is a term constructing a CM commitment matrix, then we want to construct a translator f satisfying: $f(S) = T$. Given this formulation, one would expect that standard equational reasoning would apply to solve for f. Instead we found it more effective to interleave the basic steps of higher-order matching and the application of domain-specific laws.

The glue-code subgroup of Miata (Mark Burstein, Drew McDermott, Doug Smith and Stephen Westfold) investigated glue-code synthesis and produced several publications [3],[2]. At Kestrel we implemented a higher-order matcher and a simple version of the glue-code generator.

3.5. MIATA Technology Integration Experiment

Our participation in the MIATA TIE initially suggested the need to focus on glue-code synthesis. We implemented a higher-order matcher and a version of the glue-code generator that runs on the Specware 2000 system (which runs on Windows, Linux, and Solaris platforms). We ran numerous examples through the generator, leading to improvements in the higher-order matcher and the tactical control of the generator. Dr Westfold demonstrated the glue-code generator as part of the Miata TIE during the August 2000 COABS meeting.

4. Synthesis of Glue Code

Glue-code addresses the problem of getting agents to communicate with each other. By ``agent'' we mean programs that operate at a high enough semantic level that they can form new connections to other programs in order to get a job done. To make such a connection, an agent must find other agents that might carry out a task on its behalf, and then establish a dialogue with them. Several researchers have examined facets of this interchange, including how agents might search for each other [19], how they might communicate once they have linked [12], and what ``speech acts'' they might employ [7]. However, the most pressing problem is getting them to *speak the same language*. This is our focus here.

Suppose one agent, *A*, needs a certain fact, and agent *B* can supply it. Assuming that some previous ``brokering'' or ``advertising'' phase has brought the two agents together, there remains the problem that the way *A* represents facts and the way *B* represents them are probably not compatible. It is necessary to interpose a translation program between the two. We call this *glue code*. The problem is to generate glue code automatically.

For example, suppose we are given a scheduling agent *S* that produces an airlift schedule. For each aircraft, the schedule gives the sequence of flights that it is scheduled to make. We also have an agent *R* that maintains what is known as a ``commitment matrix,'' a table that specifies, for each time slot, the number of each type of aircraft that are committed to scheduled flights (i.e. not free for allocation) in that time slot. The problem is to derive a translator *f* from schedules to commitment matrices, so that *R* is able to accept the information derived from the scheduler according to its declared input specification format (API).

Some commonly considered approaches to this problem are to:

- Engineer the agents to be compatible in advance by changing one or the other to accept or generate the form required by the other.
- Attempt to develop a ``general purpose'' translation agent that will convert all messages that can be produced by agents using one semantic model or ontology into equivalent messages using a second ontology, to the extent that translations between those ontologies are well defined.

We are developing a third approach, namely, to submit the specification of the source and target messages to an agent that produces a very specific translator for that purpose. This has the potential advantage of being much more efficient if similar messages will be sent frequently once the agents have been ``introduced'', or when the data to be passed in a single message contains a large number of similar forms.

This approach to the problem can be considered, as a form of the *automatic programming* problem, but one that we believe is simpler than the general case. It ``feels like'' an exercise in moving data around, with a bit of condensing, summarization, and totaling thrown in. On the other hand, problems like this one are not trivial. The reader may wish to stop and try to produce the glue code for *S* and *R* by hand.

In what follows, we will describe our framework, and illustrate with several examples. Although, we do not have a full implementation of our approach as yet, we have developed several prototypes that can handle a variety of examples like those described in this paper. At the end of the paper, we will talk about opportunities and challenges in automating the process more fully, and in applying it in a distributed agent environment.

In addition to the agent-communication work we mentioned at the outset, much work has been done by the database community on the problem of translating between databases, where it is called the problem of *schema integration*, with subproblems of *query translation* and *value translation*. [14],[10],[8],[13]. The main differences are:

1. Database researchers assume that the main problem is to translate queries (and their results) from one formalism to another. Queries are written in a standard language such as SQL [9], and the only issue is how the relation and argument names are mapped. We want to be able to translate an arbitrary formula (or functional expression) from one formalism to the other.
2. Database researchers assume that the results of a query are tables in a standard format. Hence if you can find a translation of a query you automatically can translate the result. We will tackle the more general case of automatically generating data-structure-translation code given expressions that describe what one agent wants and what another can produce.

4.1. The Setting

The common theory of the application domain provides symbols for the concepts, operations, and properties, relationships, etc. in the domain. Its axioms constrain the meaning of the vocabulary. For example, suppose that we have an abstract database of persons $P : set(Person)$ where each $Person$ has a $name$: $Person \rightarrow string$, $id : Person \rightarrow nat$, age: $Person \rightarrow nat$, and other attributes.

Our example supposes that this abstract database has two somewhat different realizations: S and T. To express the realization relation it is convenient to use the following notation which lifts value tupling to function tupling: for f_1: $A \rightarrow B_1,\ldots,f_n$: $A \rightarrow B_{\mathrm{n}}$, the function $\langle f_1,\ldots,f_n \rangle$: $A \rightarrow B_1 \times \ldots \times B_n$ satisfies the *tupling-reduction* law

$$\langle f_1,\ldots,f_n \rangle\,(a) = \langle f_1(a),\ldots,f_n(a) \rangle$$

Using the function tupling notation, the source database S is

$$S = image(\langle name, id, age\rangle\,, P)$$

and the target database that we want to build from S is

$$T = image(\ \langle\ name, \lambda(pv)\ if\ age(pv) > 30\ then\ id(pv)\ else\ 0\rangle\,, P)$$

Formally, we want to translate from dataset S and dataset T (without reference to P) by solving the higher-order equation $f(S){=}T$.

4.2. Inference Rules

The following rules correspond to the basic steps of a second-order matching algorithm [11].

Imitation Rule
for any $f : A \rightarrow C$
and g: $B_1 \times \ldots \times B_n \rightarrow C$ where B_i is not a function type

and $h_i : A \rightarrow B_{\mathrm{i}}$ for $1 \leq i \leq n$,

$$f(u) = g(v_1,\ldots, v_n)$$

$$\Longleftarrow$$

$$f(x) = g(\ldots, h_i(x),\ldots) \wedge h_i(u) = v_i \text{ for } 1 \leq i \leq n$$

Projection Rule
for any $f : A_1 \times\ldots\times A_m \rightarrow C$
and g: $B_1\times\ldots\times B_n \rightarrow A_i$ for some i, $1 \leq i \leq m$,

$$f(a_1,\ldots, a_m) = \mathrm{g}(b_1,\ldots, b_n)$$

$$\Longleftarrow$$

$$f(x_1,\ldots, x_m) = x_i \wedge a_i = \mathrm{g}(b_1,\ldots, b_n)$$

The terms ``imitation" and ``projection" are standard in the matching and unification literature. Here we formulate them as inference rules to be used in a backward inference mode. It can be easily seen that they follow from universal instantiation and equality substitution rules. We also use the usual rules for handling equalities and equivalences, and the basic rules of the lambda calculus: α, β, and η-reduction. The following law is useful for the backward chaining-style of proof that we adopt.

Image Decomposition Law
for any $g : A \rightarrow B$,
and $h : A \rightarrow C$,
and f : set(B) → set(C),
and $i : B \rightarrow C$,

$$f(image(g, As)) = image(h, As)$$

$$\Longleftarrow$$

$$f(Bs) = image(i, Bs) \wedge i(g(x)) = h(x)$$

4.3. Simple Example -- Translating between Personnel Databases

Suppose that we have an agent that offers personnel database services, in particular it provides

$$S = image(\langle name, id, age\rangle , P)$$

and another agent requires

$$T = image(\langle name, \lambda(pv)\ if\ age(pv) > 30\ then\ id(pv)\ else\ 0\rangle , P)$$

The problem is to calculate f such that $f(S) = T$:

$$f(image(\langle name, id, age\rangle , P)) = image(\langle name, \lambda(pv)\ if\ age(pv) > 30 \text{ then } id(pv) \text{ else } 0\rangle , P)$$

$\Longleftarrow$ using Image Decomposition with
$\{g \longmapsto \langle name, id, age\rangle$,
$h \longmapsto \langle name, \lambda(pv)\ if\ age(pv) > 30 \text{ then } id(pv) \text{ else } 0\rangle$

$f(X) = image\ (i, X)$
$\wedge\ i(\ \langle name, id, age\rangle\ (p)) = \langle\ name, \lambda(pv)\ if\ age(pv) > 30\ then\ id(pv)\ else\ 0\rangle\ (p)$

$\Longleftrightarrow$ The first conjunct ion provides a substitution/definition for f, and tupling-reduction is applied to the second conjunct

$i(name(p), id(p), age(p)) = \langle name(p), if\ age(p) > 30\ then\ id(p)\ else\ 0\ \rangle$

$\Longleftarrow$ Imitation with $\{g \longmapsto \lambda\ (x, y)\ \langle x, y\rangle\}$

$i(n, i, a) = \langle\ i_1\ (n, i, a),\ \ i_2\ (n, i, a)\ \rangle$
$\wedge\ \ i_1\ (name(p), id(p), age(p)) = name(p)$
$\wedge\ \ i_2\ (name(p), id(p), age(p)) = if\ age(p) > 30\ then\ id(p)\ else\ 0$

Again, the first conjunct ion provides a substitution/definition for i, and projection solves for i_1, Imitation for i_2

$i(n, i, a) = n\ \Lambda\ name(p) = name(p)$
$\wedge\ i_2\ (n, i, a) = if\ i_{21}\ (n, i, a)\ \ then\ i_{22}\ (n, i, a)\ \mathrm{else}\ i_{23}\ (n, i, a)$
$\wedge\ \ i_{21}\ (name(p), id(p), age(p)) = age(p) > 30$
$\wedge\ \ i_{22}\ (name(p), id(p), age(p)) = id(p)$
$\wedge\ \ i_{23}\ (name(p), id(p), age(p)) = 0$

The rest of the derivation is straightforward application of imitation and projection. Summing up, we have constructed the functions

$f(X) =\ image(\ i, X)$
$i(n, i, a) = \langle\ i_1\ (n, i, a), i_2\ (n, i, a)\ \rangle$
$i_1\ (n, i, a) = n$
$i_2\ (n, i, a) = if\ i_{21}\ (n, i, a)\ \ then\ i_{22}\ (n, i, a)\ \ else\ i_{23}\ (n, i, a)$

After unfolding the definitions below f, we get the translation code

$$T =\ f\ (S) = image(\lambda(\langle n, i, a\rangle\ )\ \ \langle n, if\ a > 30\ then\ i\ else\ 0\rangle\ , S).$$

This example is typical of our derivations in that mostly matching rules are applied, with some law applications interspersed.

4.4. Summary

Other, more complex, examples that we have solved in this manner include

- translating an aircraft schedule to a commitment matrix
- translate a mission database to a list of flights for each aircraft

- given flight data and cargo data, construct a manifest list for each flight
- translating from cargo weight expressed in terms of cargo classes to cargo weight expressed in terms of individual objects (this problem addressed the issue of conceptual mismatch and possibly conflicting assumptions).

These examples, and many others, are currently working in the glue-code generator built by Dr. Westfold in the Specware/Designware system [18],[16].

5. Publications Resulting from this Project

Lee Blaine, Limei Gilham, Junbo Liu, Douglas R. Smith, and Stephen Westfold, Planware -- Domain-Specific Synthesis of High-performance Schedulers, Proceedings of the Thirteenth Automated Software Engineering Conference, IEEE Computer Society Press, Los Alamitos, California, October, 1998, 270--280.

Smith, D.R., Toward a Theory of Specware Specs, Kestrel Institute Technical Report, July 1999.

Smith, D.R., Subsort Introduction, Kestrel Institute Technical Report, October 1999.

Smith, D.R., Mechanizing the Development of Software, invited paper in Calculational System Design, Proceedings of the NATO Advanced *Study Institute*, Eds. M. Broy and R. Steinbrueggen, IOS Press, Amsterdam, 1999, 251-292.

Smith, D.R., Designware: Software Development by Refinement, invited paper in *Proceedings of the Eighth International Conference on Category Theory and Computer Science*, Edinburgh, September, 1999.

Nancy Durgin and John C. Mitchell and Dusko Pavlovic, Protocol composition and correctness, Kestrel Institute Technical Report, January, 2000.

M. Burstein, D. McDermott, D.R. Smith, and S.J. Westfold, Formal Derivation of Agent Interoperation Code, Proceedings of the Formal Approaches to Agent-Based Systems Workshop, NASA Goddard Space Flight Center, MD, April 2000.

M. Burstein, D. McDermott, D.R. Smith, and S.J. Westfold, Derivation of Glue Code for Agent Interoperation, invited paper in Journal of Autonomous Agents and Multi-Agent Systems, 2001 (earlier version in Proceedings of the Agents 2000 Conference, Barcelona, Spain, May 2000).

Nancy Durgin and John C. Mitchell and Dusko Pavlovic, A compositional logic for protocol correctness, in *Proceedings of Computer Security Foundations Workshop 2001*, Ed.S. Schneider, 2001 (ftp://ftp.kestrel.edu/pub/papers/pavlovic/CLPC.ps.gz).

Westfold, S.J. and Smith, D.R., Synthesis of Efficient Constraint Satisfaction Programs, Knowledge Engineering Review 16(1), Special Issue on AI and OR, 2001, 69-84.

D. McDermott, M. Burstein, and D.R. Smith, Overcoming Ontology Mismatches in Transactions with Self-Describing Service Agents, in Proceedings of the First Semantic Web Working Symposium (SWWS '01), 30 July - 1 August 2001.

6. References

[1] Blaine, L., *et al.* Planware - Domain-Specific Synthesis of High-Performance Schedulers. In, *Proceedings of the Thirteenth IEEE International Automated Software Engineering Conference (ASE 1998)*. Hawaii, October 13-16, 1998. IEEE.

[2] Burstein, M., D. McDermott, and D. Smith. Overcoming ontology mismatches in transactions with self-describing service agents. In, *Proceedings of the First Semantic Web Working Symposium (SWWS '01)*. Stanford University, California, 2001.

[3] Burstein, M., *et al.* Formal derivation of agent interoperation code. *Journal of Autonomous Agents and Multi-Agent Systems*, 2001.

[4] Durgin, N., J.C. Mitchell, and D. Pavlovic. A compositional logic for protocol correctness. In, *Proceedings of Computer Security Foundations Workshop*, 2001.

[5] Durgin, N., J.C. Mitchell, and D. Pavlovic. *Protocol composition and correctness*. Kestrel Institute Tech. Rep. KES.U.00.01, January 2000.

[6] Emerson, T. and M. Burstein. Development of a constraint-based airlift scheduler by program synthesis from formal specifications. In, *Proceedings of the Fourteenth Automated Software Engineering Conference*, October 1999. IEEE Computer Society Press.

[7] Finin, T., Y. Labrou, and J. Mayfield. Kqml as an agent communication language. In, *Software Agents*. : AAAI Press/MIT Press, 1997.

[8] Florescu, D., L. Raschid, and P. Valduriez. A Methodology for Query Reformulation in C using semantic knowledge. *Int. J. of Cooperative Information Systems*, 1996.

[9] Groff, J.R. and P.N. Weinberg. *The Complete Reference SQL*: McGraw-Hill, 1998.

[10] Hammer, J., *et al.* Extracting semistructured information from the web. In, *Workshop on Management of Semistructured Data*. Tucson, Arizona, 1997.

[11] Huet, G. and B. Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*. 11, 1978, pp. 31--55.

[12] Martin, D.L., A.J. Cheyer, and D.B. Moran. The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, 1999, pp. 91--128.

[13] Milo, T. and S. Zokar. Using schema matching to simplify heterogeneous data translation. In, *Proc. Conf. on Very Large Data Bases*, 1998, pp. 122--133.

[14] Papakonstantinou, Y., *et al.* A query translation scheme for rapid implementation of wrappers. In, *Proc. DOOD'95*, 1995.

[15] Rehof, J. and T. Mogenson. Tractable constraints in finite semilattices. *Science of Computer Programming*. 35, 1999, pp. 191--221.

[16] Smith, D.R. Mechanizing the development of software. In, *Calculational System Design, Proceedings of the NATO Advanced Study Institute*, M.Broy and R. Steinbrueggen, Eds. Amsterdam: IOS Press, 1999, pp. 251--292.

[17] Smith, D.R. *Toward a theory of Specware specs*. Kestrel Institute Tech. Rep., July 1999.

[18] Srinivas, Y.V. and R. Jüllig. Specware™: Formal Support for Composing Software. In, *Proceedings of the Conference on Mathematics of Program Construction*, B. Moeller, Ed. Berlin: Springer-Verlag, 1995, pp. 399--422. Lecture Notes in Computer Science, Vol. 947.

[19] Sycara, K., *et al.* Dynamic service matchmaking among agents in open information environments. *ACM SIGMOD Record - Special Issue on Semantic Interoperability in Global Information Systems*. 28(1), 1999, pp. 47--53.

[20] Westfold, S. and D. Smith. Synthesis of efficient constraint satisfaction programs. *Knowledge Engineering Review*. 16(11), 2001, pp. 69--84.